VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Agregace zpráv z internetu

Internet News Aggregation

2020 Petr Faruzel

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Petr Faruzel**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Agregace zpráv z internetu
Internet News Aggregation

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit systém pro agregaci novinových zpráv z internetu. Systém bude umožňovat přidávat zdroje dat, pro každý zdroj pak samostatný modul pro extrakci textu a dalších částí. Primárně bude směřován na veřejné zdroje dat, které nejsou ukryty za Paywallem.

Práce bude obsahovat:
1. Popis existujících řešení.
2. Návrh systému a jeho jednotlivých částí.
3. Implementace systému.
4. Otestování systému za delší časové období na minimálně 20 zdrojích dat.

Seznam doporučené odborné literatury:

[1] Allen B. Downey: Think Python: How to Think Like a Computer Scientist, 2nd edition, O'Reilly, 2015.
[2] David Beazley & Brian K. Jones: Python Cookbook, O'Reilly, 3rd edition, 2013.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Jan Platoš, Ph.D.**

Datum zadání: 01.09.2019
Datum odevzdání: 30.04.2020

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 5. května 2020 .....

Rád bych poděkoval svému vedoucímu bakalářské práce, panu doc. Ing. Janu Platošovi, Ph.D. za cenné rady při vývoji softwaru a psaní této práce.

Abstrakt

Cílem této bakalářské práce je prostudovat aktuálně dostupné způsoby agregace zpráv z různých internetových zdrojů. Poté pomocí těchto znalostí vytvořit vlastní systém v jazyce Python, který bude tyto zprávy automaticky stahovat z předem definovaných webových stránek a následně z nich extrahuje text a další relevantní části. Systém posléze porovná získaná data a pokusí se najít články stejného či podobného tématu.

Klíčová slova: Agregace zpráv, Python, Django, RSS

Abstract

The purpose of this bachelor thesis is to study currently available ways of news aggregation from various internet sources. Then use this knowledge to create a custom system in Python, which will automatically download these internet news from predefined web pages and then extracts text and other relevant parts. System then compares the obtained data and tries to find articles of the same or similar topic.

Keywords: News aggregation, Python, Django, RSS

Obsah

Seznam použitých zkratek a symbolů **8**

Seznam obrázků **9**

Seznam tabulek **10**

Seznam výpisů zdrojového kódu **11**

1 Úvod **12**

2 Co je to agregátor zpráv? **13**
2.1 Typy agregátorů 13
2.2 Agregátory v praxi 14

3 Technologie **15**
3.1 Python 15
3.2 Django 15
3.3 SQL 17
3.4 RSS 18

4 Analýza **20**
4.1 Struktura projektu 20
4.2 Způsob stahování zpráv 21
4.3 Datová struktura 23

5 Implementace **25**
5.1 Vytvoření projektu 25
5.2 Tvorba grafického rozhraní 25
5.3 Realizace modelů 26
5.4 Přidávání zdrojů a filtrů 26
5.5 Tvorba modulu pro stahování dat 29
5.6 Metoda parsování značkovacích jazyků 29
5.7 Stahování zpráv 30
5.8 Zobrazení zpráv 33
5.9 Články s podobným obsahem 34
5.10 Paralelizace 35
5.11 Výkon aplikace 36

6 Závěr **38**

Literatura **39**

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript and XML
AI	–	Artificial Intelligence
BLOB	–	Binary Large Object
CSS	–	Cascading Style Sheets
HTML	–	Hypertext Markup Language
HTTP	–	Hypertext Transfer Protocol
JS	–	Javascript
MVC	–	Model View Controller
OGP	–	Open Graph Protocol
RAKE	–	Rapid Automatic Keyword Extraction
RSS	–	Really Simple Syndication
SQL	–	Structured Query Language
TLD	–	Top Level Domain
UI	–	User Interface
URL	–	Uniform Resource Locator
XML	–	Extensible Markup Language

Seznam obrázků

1 Architektura MVC 16
2 Koncept grafického designu 21
3 Diagram procesu agregace zpráv 23
4 Databázový model 24
5 Spuštění Django projektu ve virtuálním prostředí 25
6 Syntaxe pravidel pro filtrování 28
7 Seznam nejnovějších zpráv na hlavní stránce 34
8 Stránka se článkem a nabídkou podobného obsahu 35

Seznam tabulek

1 Využité internetové zdroje zpráv 27
2 Rychlost stahování s využitím SQLite 36
3 Rychlost stahování s využitím PostgreSQL 36
4 Rozdíl velikosti filtrovaných a nefiltrovaných dat 37

Seznam výpisů zdrojového kódu

1 Ukázka Django HTML šablony 16
2 Ukázka RSS kanálu obsahující jednu zprávu 19
3 Model definující ukládaná data každého článku 26
4 Stažení webové stránky pomocí knihovny requests 29
5 Ukázka parsování dokumentů pomocí knihovny bs4 a lxml 30
6 Ukázka stažení webové stránky a získání náhledového obrázku 31
7 Kód řešící extrakci nežádoucích elementů 32
8 Metoda pro extrakci klíčových slov pomocí knihovny rake-ntlk 33

1 Úvod

Celý svět je dnes propojen pomocí internetu a počet informací, které se dokáží v rámci několika okamžiků dostat až na druhou stranu planety, je enormně velký. Lidé jsou tedy již zvyklí dostávat nejnovější informace ze všech koutů světa navštívením několika webových stránek, obvykle určených pro předávání aktuálních informací či novinek. Avšak vzhledem k rychle rostoucí velikosti samotného internetu je přirozené, že vznikne i veliký počet takovýchto stránek a lidé mohou začít být lehce frustrovaní přepínáním mezi jednotlivými weby pro získání informací, které požadují.

Cílem této bakalářské práce je vytvořit informační systém, který dokáže shlukovat internetové zprávy z různých webových zdrojů pomocí RSS kanálů do jediné webové stránky a následně z nich pomocí nastavitelných modulů extrahovat relevantní části určené pro oči čtenáře. Systém poté dokáže porovnat zprávu s ostatními a nabídnout další s podobným obsahem. K implementaci tohoto systému bude využit jazyk Python a webový framework Django.

2 Co je to agregátor zpráv?

V dnešním rychle měnícím se světě s rychlým nárůstem informací začala spousta lidí hledat své útočiště právě v agregačních službách. Agregátory zpráv umožňují jednotlivcům i korporacím rychle a efektivně analyzovat a zpracovávat velké množství dat pomocí jediné aplikace. Zjednodušeně by se tedy dalo říct, že se jedná o jakýsi systém, který shlukuje zprávy z různých internetových portálů do jednoho jediného místa, kde jsou následně zobrazeny koncovému uživateli v určité předvídatelné formě [1].

Spousta agregátorů nabízí i jakýsi druh přidané hodnoty, ať už se jedná o vlastní přidávání zdrojů, definování různých kategorií či personalizaci vzhledu. Takovýto agregátor však není autorem žádné zprávy a nevytváří si žádný vlastní obsah, pouze jej stahuje a následně s ním jakýmsi způsobem pracuje. V historii by si člověk pod něčím jako je agregátor zpráv mohl představit například trafiku, kde měl na výběr z určitého počtu různých novin, přičemž se každá mohla zaměřovat na jiné téma zpráv, ať už to byly zprávy sportovní, ekonomické či ze světa vědy.

2.1 Typy agregátorů

Ačkoliv je základní myšlenka agregátorů zpráv ve své podstatě vždy stejná, tak se v průběhu času změnily trendy v technologiích, na kterých jsou tyto systémy postaveny. Z technického pohledu tedy existuje několik typů agregátorů, které je možné využívat. Prvním typem je desktopová aplikace, tedy spustitelný program, který běží na operačním systému koncového uživatele. Takovéto programy jsou však obvykle určené pouze pro jeden typ operačního systému a vývoj na vícero platformách může být náročný a drahý. Potřeba instalace takovéhoto softwaru může uživatele taktéž odradit od jeho využívání. Tyto problémy však řeší další typ agregátoru, a to agregátor webový.

Bezkonkurenční výhoda webových agregátorů je ta, že jsou dostupné odkudkoliv s přístupem k internetu, přičemž mají pouze jedinou podmínku pro jejich využívání, a to nainstalovaný webový prohlížeč. Jedná se tedy zároveň i o multiplatformní software. Webové agregátory mají i další výhodu v tom, že není potřeba žádné explicitní instalace. Na trhu se momentálně nachází několik velkých hráčů s velmi sofistikovaným systémem a komplexními algoritmy, které mohou uživatelé jejich služeb využívat.

2.1.1 Google News

Tato služba je, jak již název napovídá, vyvíjena společností Google. Svou existenci započala v prosinci 2001, kdy jej za jediný víkend vytvořil mladý inženýr frustrovaný vyhledáváním zpráv po atentátu dvou věží dne 11. září 2001. Oficiální spuštění bylo až roce 2002, přičemž až v roce 2006 se dostala z beta verze [2]. Dnes tato webová agregační služba nabízí široké spektrum zdrojů, ať už se jedná o zahraniční či české. Uživatel si může vybrat z několika témat, která

jej zajímají a algoritmy už najdou ten nejrelevantnější obsah podle předchozích vyhledávání a lokality, a to vše zcela zdarma.

2.1.2 Feedly

Jedná se o agregátor zpráv vytvořený společností DevHD. Tato služba má však oproti Google News lehce odlišnou filozofii. Je více přizpůsobitelná uživateli, a to tím způsobem, že si může nastavit zdroje zpráv dle vlastního výběru a do vlastnoručně vytvořených skupin. Oproti konkurenci tedy nenabízí mainstreamové zprávy dle lokality. Jedná se zároveň o placenou službu, kterou lze používat zdarma, avšak pouze s minimální funkcionalitou.

2.2 Agregátory v praxi

I když může být agregátor zpráv pro některé uživatele skvělým pomocníkem do každodenního života, tak se zde nachází taktéž druhá strana mince, a to samotní autoři všech těchto stahovaných zpráv. Ti mohou začít uplatňovat své autorské právo a takovéto agregátory žalovat za kradení svého obsahu. Existuje tedy několik striktních zákonů, které takovémuto nežádoucímu šíření zpráv zabraňují a dnešní online dostupné agregátory mohou zobrazovat nanejvýše titulek, krátký popisek článku a náhledový obrázek [3].

To však neznamená, že agregátory zprávy nestahují, ba naopak. Pokud chce takovýto agregační systém nabízet uživatelům relevantní obsah, je potřeba tyto zprávy stáhnout a pokusit se z nich získat veškeré potřebné informace, které by jinak nebyly dostupné. Algoritmy daných systémů si pak mohou tato data interpretovat různými způsoby a následně vybrat zprávy co nejrelevantnější pro koncového uživatele. Komplexnost takovýchto algoritmů může být velmi různorodá, od hledání podobných slov po rozeznání, zdali se slovem „Apple“ myslí jablko, nebo americká firma. Toto vše se ale odehrává pouze na pozadí a v konečné situaci je uživatelům zobrazeno pouhé zákonem povolené minimum.

3 Technologie

Tato kapitola je věnována popisu klíčových programovacích jazyků a technologií, které byly při implementaci systému pro agregaci zpráv využity a částečně také odůvodňuje, proč jsem tyto konkrétní technologie zvolil a jaké výhody nabízí.

3.1 Python

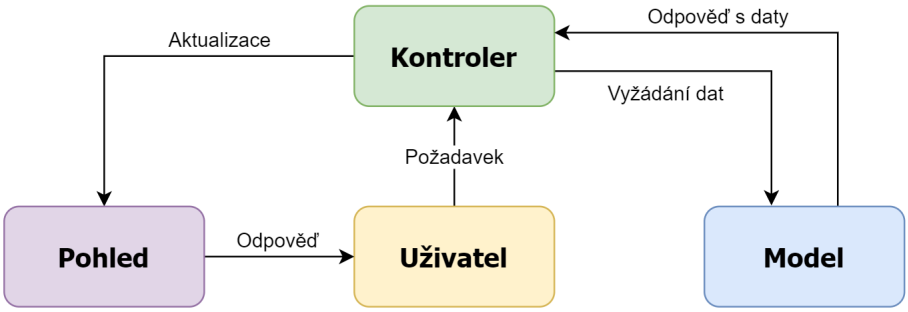
Python je interpretovaný a objektově orientovaný programovací jazyk [4]. Jeho velkou výhodou je víceúčelovost a platformní nezávislost. Filozofie tohoto jazyka je soustředěna na jednoduchou syntaxi a čitelnost kódu. Vývoj aplikace je tedy obvykle podstatně rychlejší než u jazyků, jako je například C++. Cena za rychle napsaný kód se však promítá v rychlosti provádění samotného kódu programu. Vzhledem k tomu, že se jedná o interpretovaný jazyk, tak jeho kód není možné plně kompilovat a všechny instrukce jsou vykonávány interpretem Pythonu až v době, kdy jsou zapotřebí.

Díky technologickému vývoji a moderním počítačům si však můžeme dovolit využívat o něco pomalejšího běhu programu. Aktuální výpočetní výkon běžných počítačů bývá natolik velký, že je rozdíl v rychlosti provádění několika jednoduchých operací poměrně zanedbatelný. Větší problémy nastávají až při složitějších matematických výpočtech, u kterých je toto zpomalení poznat. Python má však jednu obří výhodu, a to je ta, že můžeme urychlit výpočty využitím matematických modulů psaných v C/C++.

3.2 Django

Django je webový aplikační framework napsaný v jazyce Python, který se řadí pod otevřený software, anglicky open source. Tento framework byl veřejně prohlášen za otevřený v červenci roku 2005 a dnes je pod křídly neziskové organizace Django Software Foundation, která vytváří aktualizace a spravuje tento systém dodnes [5].

Jeho hlavním cílem je ulehčit programátorům vývoj komplexních webových stránek a zjednodušit proces vytváření a spravování databází. Jeho silnou stránkou je velmi přehledná dokumentace, na které se podílí samotná komunita a je tedy lehce pochopitelná i pro úplné začátečníky. Co se týče softwarové architektury, tak tento framework využívá konceptu MVC. Ten rozděluje datový model, uživatelské rozhraní a logiku aplikace do tří samostatných komponent. Komunikaci mezi jednotlivými komponentami lze pozorovat na obrázku 1. O veškeré UI, které uživatel vidí se stará komponenta pohledu. Jakákoliv interakce s tímto rozhraním je následně zpracovávána kontrolerem. Kontroler pak může na uživatele reagovat buď přímo, tedy aktualizuje pohled okamžitě, nebo si nejprve vyžádá data modelu. Model je komponenta, která je zodpovědná za držení dat a jejich manipulaci. Kontroler tedy přikazuje modelu, co se s daty má stát a následně je může pohledu předávat.



Obrázek 1: Architektura MVC

3.2.1 Systém šablon

Django také obsahuje několik velmi důležitých komponent. První takovou komponentou je systém šablon. Ten umožňuje přidávat dynamické prvky a proměnné do statického HTML souboru, a to pomocí speciálních značek. Tyto značky jsou následně při odpovědi na HTTP požadavek zaměněny za dynamické hodnoty. Příkladem takové situace může být zobrazení věku uživatele v šabloně výpisu. Logika systému vypočítá aktuální věk a následně tuto vypočtenou hodnotu dosadí do místa specifikovaného značkou, přičemž koncový uživatel, který požadavek poslal, uvidí pouze vypočtené číslo. Systém značek taktéž umožňuje přistupovat k proměnným různých objektů, využívat cykly a vytvářet podmínky. Takovýto systém tedy velmi zjednodušuje práci a poskytuje obrovskou výhodu při vytváření komplexních webových stránek.

Další skvělou vlastností tohoto systému je možnost definování několika bloků uvnitř jednotlivých šablon. Každý blok definuje místo, do kterého je možné vložit šablonu jinou. Tím se tedy vytváří určitá hierarchie, pomocí které je možné využít obsah potomka a vložit jej do předem definovaného bloku v rodičovské šabloně. Typickým příkladem je vkládání různého obsahu do rodičovské šablony, která obsahuje na každé stránce stejné záhlaví a zápatí. Tyto elementy pak není potřeba při každé šabloně opisovat, ale stačí pouze využít jejich implementaci v rodičovské šabloně.

```
<div id="zahlavi">Autor teto stranky ma {{ vek }} let.</div>
<div id="obsah-stranky">
    {% block obsah %} <!-- Místo pro šablonu potomka --> {% endblock %}
</div>
<div id="zapati">...</div>
```

Výpis 1: Ukázka Django HTML šablony

3.2.2 Objektově relační mapování

Objektově relační mapování je další velmi důležitou komponentou tohoto frameworku. Django má vlastní třídu Model, kterou je možné využít a vytvářet díky ní modely vlastní. Tyto modely by se daly označit za jakési plány, anglicky blueprints, podle kterých se následně při migraci vytváří tabulky v databázi. Každý model se mapuje na samostatnou tabulku. Jednotlivé instance těchto objektů poté reprezentují jeden řádek tabulky a je nad nimi možné provádět nejběžnější operace, které jsou ekvivalentní operacím SELECT, INSERT, UPDATE a DELETE v SQL. Každá instance má mimo programátorem definovaných atributů také automaticky generovaný atribut ID, který slouží k jedinečné identifikaci objektu v databázi. Syntaxe však umožňuje modifikovat tento atribut v případě potřeby. Modely taktéž obsahují velký počet předem definovaných datových typů, od typu boolean až po samotný BLOB.

3.2.3 Nativní podpora databázových systémů

Django ve svém jádru podporuje několik základních databázových systémů [5]. Těmi jsou:

1. PostgreSQL
2. MariaDB
3. MySQL
4. Oracle
5. SQLite

Každá z těchto databází oplývá několika cennými výhodami, stejně jako i slabými stránkami. Django při prvotním vytvoření projektu pracuje s databází SQLite, která bývá ve fázi prototypování obvykle dostačující. Při mém bádání o těchto databázových systémech mě však taktéž zaujala databáze PostgreSQL. Dovolím si tedy v sekci SQL popsat klíčové vlastnosti těchto databází včetně rozdílu mezi nimi a vysvětlit důvod, proč jsem se rozhodl právě pro SQLite

3.3 SQL

SQL je standardizovaný strukturovaný dotazovací jazyk, který se používá pro správu a organizování dat v relačních databázích [6]. Ačkoliv se jazyku říká dotazovací, tak je tento název pouze stručný, nikoli zcela výstižný. SQL není jen dotazovací jazyk, protože můžeme s jeho pomocí definovat data, resp. strukturu tabulek, naplňovat sloupce tabulky daty a definovat vztahy mezi jednotlivými položkami dat. Umožňuje taktéž řízení přístupových oprávnění na různých úrovních, čímž chrání data před náhodným nebo úmyslným poškozením dat.

3.3.1 SQLite

SQlite je SQL databázový engine napsaný v jazyce C. Jedná se zároveň o velmi malý, jednoduchý a multiplatformní software, který je nejběžněji využívaný pro přenos obsahu mezi systémy a dlouhodobou archivaci dat [7]. Je reprezentován jediným souborem, typicky s příponou .sqlite3, který obsahuje veškerá data. Oproti jiným databázovým systémům tedy nevyžaduje databázový server a funguje pouze jako samostatný soubor na disku. Tím se mnohonásobně usnadňuje manipulace při přenosu dat, avšak za cenu nižší rychlosti práce s daty, která by však v mém případě prototypování nebyla nijak znát. Možnost využití této databáze na vícero platformách by pro mě bylo taktéž velmi užitečné, jelikož bych mohl jednoduše využít svůj projekt napsaný v systému Windows a přenést jej na svůj Raspberry Pi běžící na systému Raspbian. Systém by na něm tedy mohl zprávy agregovat nepřetržitě a veškerá data bych pak měl v jediném souboru, který bych si mohl přetáhnout zpátky na systém Windows pro podrobnější analýzu a prezentaci svého postupu při vývoji.

3.3.2 PostgreSQL

PostgreSQL je objektově relační databázový systém vyvíjen primárně společností PostgreSQL Global Development Group, která jej publikovala jakožto otevřený software, díky čemuž se na jejím vývoji dnes podílí i velká řada developerů a firem [8]. Tato databáze je opravdu mocný nástroj, který nezřídka využívají i velké korporátní společnosti. Jedná se taktéž o multiplatformní software, který je však tvořen primárně pro unixové systémy. Existují zde i balíčky pro Windows, avšak při pokusu o přenesení databáze na jinou distribuci mohou vznikat menší zádrhely a proces není tak jednoduchý, jako například u SQLite, jelikož celý systém potřebuje ke své funkcionalitě běžící databázový systém. Byla by tedy potřeba mít tento systém nainstalovaný na všech zařízeních, na kterých bych se svůj projekt pokoušel zprovoznit.

3.3.3 Rozdíl mezi databázemi a zhodnocení

Má volba SQLite byla tedy podpořena převážně jednoduchou přenositelností tohoto databázového systému mezi vícero operačních systémů bez potřeby instalace běžícího databázového systému. Toto rozhodnutí jsem však učinil pouze z důvodů vývoje a prototypování finálního softwaru. Pokud by měl být takovýto systém někdy v budoucnu nasazen do produkčního prostředí, bylo by nutné využít databázi typu PostgreSQL. Avšak díky možnostem Djanga by se nemusel měnit žádný kód, jelikož by stačilo pouze zaměnit databáze, provést migrace a systém by dokázal běžet bez problémů dál.

3.4 RSS

Formát RSS (Really Simple Syndication) je založený na schématu XML a slouží ke zjednodušenému čtení obsahu. Umožňuje sledování všech nejnovějších informací z různých internetových

zdrojů na jednom jediném místě [9]. Syntaxi tohoto formátu můžeme vidět ve výpisu 2. První řádek definuje typ dokumentu a jeho kódování, což je důležité pro správné načtení celého kanálu pomocí webového prohlížeče. Následuje začátek samotného RSS elementu, který obsahuje číslo své verze a jediného potomka s názvem „channel“, který v sobě skrývá všechny zbývající elementy. Prvních pár elementů v prvku „channel“ obsahuje již unikátní data a uvádí vždy informace o místě, ze kterých veškeré zprávy pochází. V případě výpisu 2 je zdrojem vymyšlená stránka „mojestranka.cz“, která je zdrojem zpráv ze světa. Veškeré zprávy jsou následně uvnitř elementu „item“. Každý takový element je povinen mít dle standardu RSS alespoň tři prvky, a to titulek, odkaz na zprávu a krátký popisek zprávy.

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
  <title>Mojestranka.cz - Zpravy ze sveta</title>
  <link>https://mojestranka.cz</link>
  <description>Popisek webu</description>
  <item>
    <title>Zprava ze sveta</title>
    <link>https://mojestranka.cz/zprava</link>
    <description>Popis zpravy ze sveta</description>
  </item>
</channel>
</rss>
```

Výpis 2: Ukázka RSS kanálu obsahující jednu zprávu

3.4.1 Historie RSS

Nebylo to dlouho po vzniku internetu, kdy začali tvůrci prvních webových stránek vymýšlet způsoby, jakými by bylo možné zajistit jakési vzájemné propojení stránek na vícero různých serverech. Z hlediska propagace bylo pro takovéto tvůrce velmi výhodné spolupracovat s jinými autory podobného obsahu, kteří na sebe mohou odkazovat a tím si na svůj web přivést mnoho nových, případně i stálých uživatelů.

Společnost Netscape tedy roku 1999 vyvinula svou první verzi RSS 0.9. Tato technologie byla s postupem času vylepšována a vzniklo několik dalších verzí. Nejnovější verzí je dnes RSS 2.0, která spadá pod právnickou fakultu Harvardovy univerzity [1, 10].

4 Analýza

4.1 Struktura projektu

Před začátkem jakéhokoliv projektu je dobré si nejprve ujasnit, jak bude vytvářený systém fungovat na jakési vyšší úrovni abstrakce. S ohledem na dnešní vysokou popularitu webů je v mém nejlepším zájmu vytvořit celý systém ve formě webové aplikace. Rozhodl jsem se využít jazyk Python a framework Django, jelikož jsem s oběma technologiemi v minulosti již pracoval a byly pro mě tedy jasnou volbou.

4.1.1 Logická část

Struktura projektu, jako je tento, by měla být do budoucna rozšiřitelná tak, aby dalo nejen přidávat nových funkcionalit, ale aby zároveň existovala i možnost jednoduše zaměnit již existující řešení, na kterém systém běží. Bylo by tedy dobré, aby každá aplikace Djanga plnila svou vlastní nezávislou roli s minimálním ovlivněním ostatních. Rozhodl jsem se tedy, že vytvořím celkem dvě další aplikace, které budou základním kamenem celého systému.

1. **Settings** - Tato aplikace je zodpovědná za práci se systémovým nastavením. Jedná se převážně o manuální přidávání veškerých zdrojů a úpravu jejich filtrů. Aplikace je pak zároveň i zodpovědná tyto filtry správně přečíst a následně nad nimi korektně provést požadované operace.

2. **Database** - Tato aplikace se stará o veškerou manipulaci s uloženými daty, které se týkají samotných agregovaných článků. Je taktéž zodpovědná za jejich korektní stažení, správné přečtení, bezchybnou manipulaci a následné uložení.

Závislosti mezi těmito dvěma aplikacemi se sice nikdy úplně nezbavíme, avšak v budoucnu by neměl být problém nahradit jednu za jinou, třeba i s úplně rozdílnou vnitřní logikou.

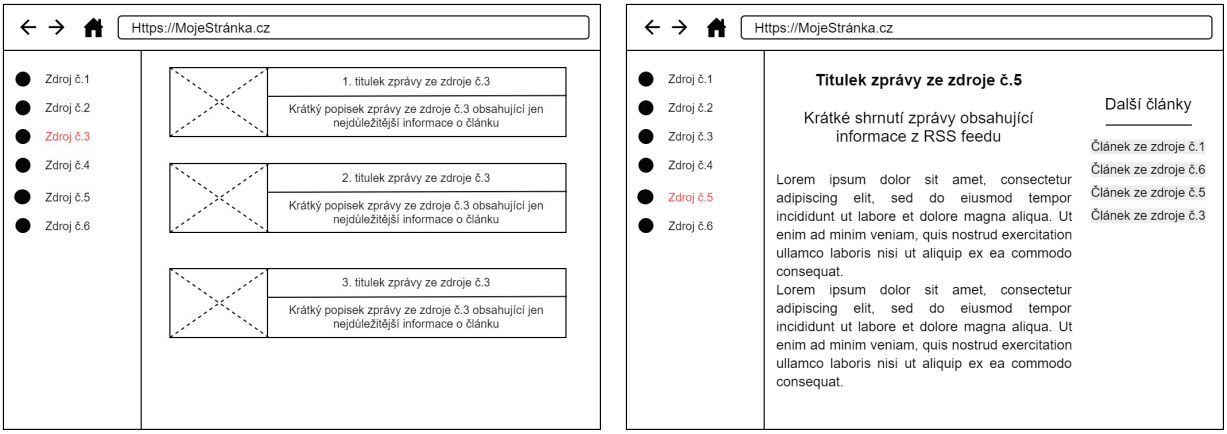
Systém se tedy bude pokoušet stahovat stránky z předem definovaných zdrojů. Tyto informace následně zpracuje do strojově čitelného formátu, který se za využití filtrů očistí od nežádoucích elementů a zbude jen výsledný text, který budeme chtít zobrazit uživateli. Při tomto procesu se zároveň bude ukládat i seznam klíčových slov, pomocí kterých se budou hledat články podobného obsahu.

4.1.2 Grafická část

Systém však nebude mít pouze logickou část, ale taktéž i část grafickou, resp. front-end. Tím, že se jedná o webovou aplikaci, bude potřeba využít HTML, CSS a JS. Práce s těmito technologiemi bez jakýchkoliv knihoven by však znamenala o něco zdlouhavější a komplikovanější proces tvorby celého UI. Rozhodl jsem se proto využít framework Bootstrap [11]. Ten je skvělý pro vytvoření základního grafického prototypu aplikace. Dá se však využít i pro komplikované aplikace

určené do produkčního prostředí. Nabízí velmi jednoduchou implementaci běžně využívaných vzhledů webových stránek a dokáže mi tedy ve velké míře ulehčit práci s designováním UI. Web také potřebuje mít určitou logiku na straně klienta. Využití Javascriptu je tedy samozřejmostí. Rozhodl jsem se využít i knihovnu JQuery [12], která mi zjednoduší dynamickou manipulaci se všemi prvky na stránce.

Má představa grafického konceptu vychází z webu Feedly a je vyobrazena na obrázku 2. Hlavní stránka bude obsahovat seznam veškerých webových zdrojů, ze kterých se zprávy budou stahovat a ze kterých si uživatel bude moci vybrat. Kliknutím na takovýto zdroj se zobrazí jeho poslední agregované zprávy. Tyto zprávy budou zobrazovány na stejné stránce vedle zmíněného seznamu a budou obsahovat náhledový obrázek, nadpis a krátký popis obsahu článku. Při kliknutí na tento náhled se následně zobrazí celá stránka se zprávou a zároveň bude vytvořen a zobrazen seznam několika nejpodobnějších článků, které bude systém schopen uživateli nabídnout.



(a) Hlavní stránka

(b) Stránka se článkem

Obrázek 2: Koncept grafického designu

Nesmíme však zapomenout na to, že bude potřeba někde pracovat i se zdroji dat a jejich nastavením. Bude zde tedy taktéž potřeba vytvořit samostatnou stránku s nastavením, na níž bude možné přidávat a odebírat tyto zdroje, a to včetně vytváření, modifikací a mazání všech jejich filtrů. Vzhledem k tomu, že nastavení by bylo v produkci dostupné pouze administrátorům a jiným oprávněným osobám, tak není potřeba vytvářet nijak sofistikované či jinak na oko pěkné UI jako u hlavní stránky. Tato stránka tedy bude klást důraz převážně na funkcionalitu.

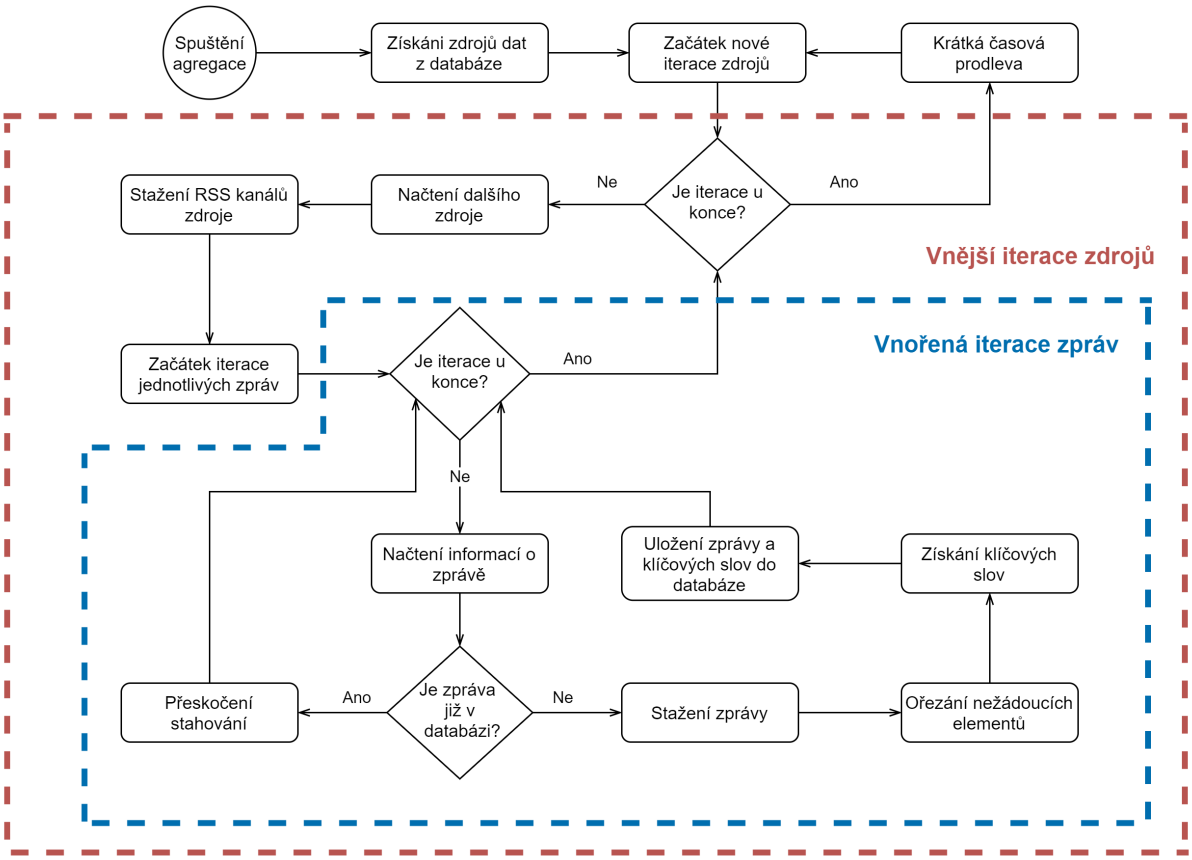
4.2 Způsob stahování zpráv

Existuje několik způsobů, jakými lze k agregaci zpráv přistupovat. Můžeme se pokusit vytvořit algoritmus, který by s minimální pomocí programátora dokázal najít prostřednictvím odkazů různé internetové portály zaměřující se na tvorbu zpráv, ze kterých by si následně dokázal sám

extrahovat všechny relevantní odkazy a vyfiltrovat z nich pouze požadovaný text. Takovýto algoritmus by byl opravdu komplexní a musel by dokázat rozeznat, co je ve skutečnosti text článku a co například pouhá reklama. Toho by šlo dosáhnout například s využitím AI, které by bylo v určité míře schopno rozpoznat text a s minimální chybovostí jej označit jako skutečný článek. Něco takového by však bylo velmi časově náročné. Já sám jsem se rozhodl pro druhý, o něco víc mechanický způsob agregace s myšlenkou, že bych následně takovýto systém mohl v budoucnu rozšířit a zvýšit jeho komplexnost vytvořením vlastní AI.

Druhý, tedy můj vybraný způsob stahování zpráv, by se dal přirovnat k prvnímu, akorát s požadavkem na určitý druh manuálního přednastavení různých kritérií před začátkem agregace. Veškeré zdroje se v tomto případě budou přidávat ručně. Algoritmus si poté pomocí RSS kanálu stránky dokáže najít požadovaný odkaz na zprávu. U každého zdroje bude pak nutno nastavit i samostatný filtr, který dokáže rozlišit, co je ve skutečnosti zpráva a odstranit veškeré nechtěné elementy, jako například reklamy. Hlavní rozdíl a nevýhoda oproti předchozímu přístupu je ta, že jsme závislí na RSS kanálu. Bez něj by začala být agregace o dost komplikovanější a jeho chybovost by pravděpodobně také vzrostla. Při tomto přístupu však můžeme s postupem času automatizovat jednotlivé komponenty a nahradit je například za již zmíněné AI. Můžeme tedy v budoucnu dosáhnout kvalit prvního zmíněného způsobu agregace.

Z hlediska implementace by měl systém stahovat zprávy v nekonečné smyčce. Základní myšlenka celého procesu agregace je znázorněna aktivitním diagramem na obrázku 3. Po spuštění stahování se z databáze načtou manuálně přidané zdroje. Následně se spustí vnější cyklus, který bude postupně všechny tyto zdroje procházet a začne stahovat jejich RSS kanály. Pro každý stažený kanál se následně spustí další, nyní již vnořený cyklus, který bude procházet všemi zprávami, které budou v právě staženém dokumentu dostupné a začne zjišťovat, zdali tyto konkrétní zprávy někdy v minulosti již nestáhl. Tohoto by šlo dosáhnout například porovnáním URL stránek, na kterých se zpráva nachází, jelikož se vlastně jedná i o jejich unikátní identifikátor. V případě, že se zpráva již v databázi nachází, tak se stahování jednoduše přeskočí a pokračuje se zprávou další. Pokud se bude jednat o zprávu novou, tak si systém stránku se článkem stáhne a pomocí filtrů získá jen ta data, která nezbytně potřebuje. Následně se z takovéto očištěné stránky načtou i klíčová slova a vše se uloží do databáze. Iterace budou probíhat do té doby, než jsou vyčerpány všechny zdroje. Po dokončení vnější iterace se následně spustí předem definovaná časová prodleva, po které se celý proces stahování zopakuje.

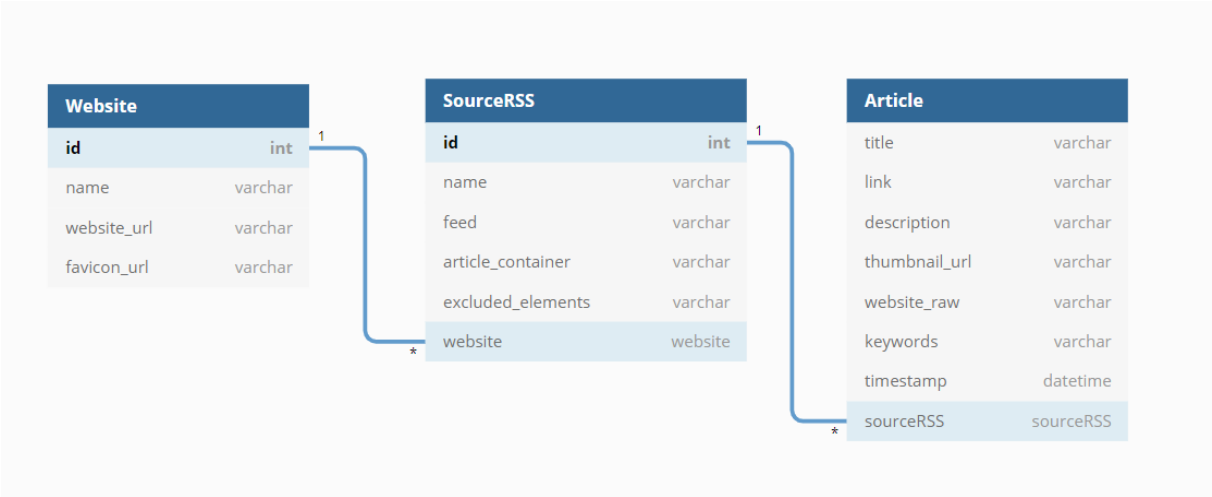


Obrázek 3: Diagram procesu agregace zpráv

4.3 Datová struktura

Klíčová položka celého systému jsou samotná data. Je důležité si určit, jaká data se vůbec budou ukládat a jakým stylem se s nimi bude pracovat. Vytvořil jsem tedy databázový model, podle kterého se budu při vytváření projektu řídit a který je znázorněn na obrázku 4. Jako základní kámen celé databázové struktury by měla posloužit tabulka s informacemi o stránkách, ze kterých se budou data stahovat. Důležitost této tabulky však nebude v jejím obsahu, ale spíše v její funkci, jelikož bude tvořit jakýsi základ celé logické hierarchie. Na ní bude následně závislá tabulka s RSS kanály. Vzhledem k tomu, že jeden web může mít i vícero různých RSS kanálů pro několik odlišných témat zpráv, tak by bylo vhodné umožnit ukládání každého takovéhoto tématu samostatně. Ke každé stránce bude díky této tabulce možné přidávat a následně samostatně spravovat několik témat, což by v budoucnu mohlo být při hledání podobných článků velmi nápomocné. Taktéž bude možné ke každému tématu přidělit samostatný a unikátní filtr. Může se totiž stát, že jeden web má různý vzhled pro vícero témat a jednotná filtrovací logika by zde byla nepoužitelná. Toto z mého pozorování sice není časté, ale narazil jsem na několik stránek, kde by byla tato funkcionalita nezbytná.

Poslední, avšak nejdůležitější tabulka, bude tabulka s informacemi o stahovaných článcích a s nimi spojenými daty. Zde se bude nacházet ořezaná stránka se článkem. Zároveň se tu vyplatí samostatně ukládat i titulek, odkaz náhledového obrázku a popisek článku, jelikož při zobrazování pouhého seznamu článků nebude potřeba tyto informace znovu hledat, a tedy se celý systém v určité míře urychlí. Dále se zde bude nacházet i informace o časovém razítku, která bude potřeba při hledání článků v určitém časovém období. Jako poslední ukládaná položka budou informace klíčových slovech, určených pro hledání podobných článků. Původně jsem přemýšlel nad možností vytvoření jedné další tabulky pouze pro klíčová slova. Nakonec jsem však od tohoto řešení ustoupil a rozhodl se využít technologie JSON. Tato slova budu tedy taktéž ukládat do tabulky obsahující článek.



Obrázek 4: Databázový model

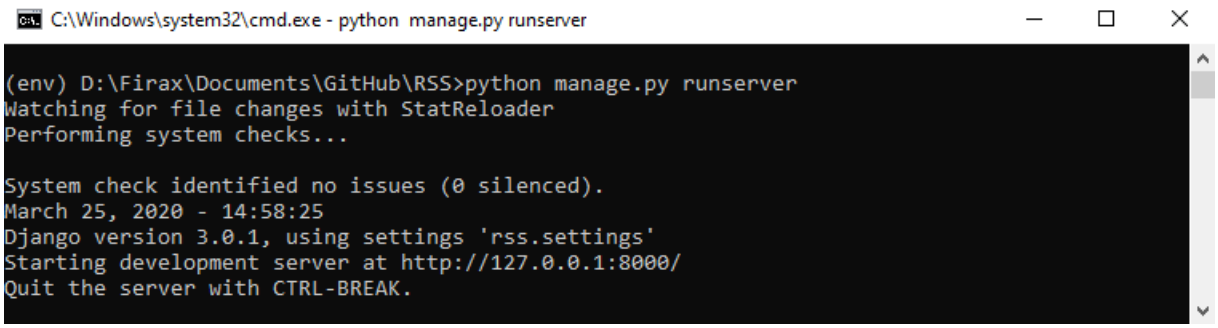
5 Implementace

Tato část je věnována popisu postupu implementace systému a všech řešení, která jsem při jeho vývoji učinil. Celou implementaci jsem pojal s myšlenkou, že vytvářím jakýsi softwarový produkt určený pro koncového uživatele, který by rád využíval webový agregátor zpráv.

5.1 Vytvoření projektu

Prvním krokem před vytvořením samotného projektu je potřeba mít nainstalovány všechny potřebné nástroje k jeho zprovoznění. Mým prvním krokem tedy byla instalace Pythonu, konkrétně 64bitové verze 3.6.0. Následně jsem si vytvořil virtuální prostředí pomocí modulu venv [4], ve kterém je projekt spouštěn. Díky tomuto prostředí mohu pak jednoduše přenést celý projekt na jiný počítač bez nutnosti instalace všech nezbytných balíčků a zároveň mám přehled o tom, které balíčky můj projekt potřebuje, jelikož je oddělen od všech mých již nainstalovaných balíčků v systému. Do tohoto prostředí jsem následně nainstaloval pomocí pipu samotný framework Django a mohl jsem začít s vytvářením nového projektu.

Vytvoření projektu pomocí Djanga se provádí příkazem **django-admin startproject *názevprojektu***. Pro své řešení jsem se rozhodl pojmenovat projekt RSS, jelikož je na této technologii celý můj systém postavený. Django poté automaticky vytvoří všechny nezbytné soubory potřebné pro spuštění webové aplikace. Pokud je projekt vytvořen, můžeme jej spustit pomocí příkazu **python manage.py runserver**. Tímto příkazem se začne vykonávat script v Pythonu, který celou webovou aplikaci spustí defaultně na adrese **127.0.0.1:8000**. Celý tento proces je možné pozorovat na obrázku 5, kde nám konzole vypisuje veškeré důležité informace. Při následném otevření této stránky se zobrazí úvodní obrazovka Djanga, díky které si můžeme být jisti, že celý projekt byl úspěšně vytvořen a je možné na něm začít pracovat.



Obrázek 5: Spuštění Django projektu ve virtuálním prostředí

5.2 Tvorba grafického rozhraní

Pro jakoukoliv interakci se samotným systémem je vhodné mít i uživatelsky přívětivé a pochopitelné UI. Začal jsem tedy tvorbou samotné HTML šablony úvodní stránky. Navigaci stránky jsem se rozhodl umístit do záhlaví, jelikož mi tato volba přišla jako nejvíc intuitivní a taktéž

i jako nejčastější řešení většiny webových stránek. Následně jsem pokračoval stylizací pomocí předem připraveného vzoru vytvořeném v draw.io, který je zobrazen na obrázku 2. Poté jsem začal vytvářet sekci s nastavením, která již začala vyžadovat větší množství interakce se systémem, a tedy i vzrostl počet kódu JS, který se o veškerou interakci stará. Zde se mi také velmi hodila knihovna Bootstrap, která mi velmi zjednodušila práci při vytváření sloupcové grafické struktury onoho nastavení. Vytváření UI pro mě však nebyl pouze statický jednorázový proces, ale spíše proces fluidní a flexibilní, kdy jsem zezačátku pouze vytvořil základní vzhledový vzor a následně jej průběžně modifikoval dle požadavků nově přidaného kódu.

5.3 Realizace modelů

Před prvotní implementací základní funkcionality byla potřeba vytvořit datové modely držící data. Využil jsem tedy své analýzy a začal s implementací modelů ekvivalentních mému databázovému modelu na obrázku 4. Vytvořil jsem dvě samostatné aplikace ve svém Django projektu, konkrétně s názvy Settings a Database. Takovéto aplikace se vytváří v konzoli, a to pomocí příkazu **python manage.py startapp *názevaplikace***. V aplikaci Settings jsem následně vytvořil modely Website a SourceRSS, které slouží k uchování informací o tom, z jakých stránek a RSS kanálů se data budou stahovat a taktéž nastavení filtrů, které se budou využívat pro extrakci textu z již zmíněných webových stránek. V aplikaci database jsem poté vytvořil poslední a nejdůležitější model s názvem Article, jehož implementaci lze vidět na výpisu 3. Tento model bude držet všechna stahovaná data. Díky tomuto rozdělení aplikací bude v budoucnu možné kdykoliv zaměnit stahovací aplikaci bez toho, aby se ztratily informace o kterémkoliv zdroji či přednastaveném filtru.

```
class Article(models.Model):
    title = models.CharField(max_length=300, default="")
    link = models.CharField(max_length=500, default="")
    description = models.CharField(max_length=2000, default="")
    thumbnail_url = models.CharField(max_length=2000, default="")
    website_raw = models.TextField(default="")
    keywords = models.TextField(default="")
    timestamp = models.DateTimeField(auto_now_add=True)
    sourceRSS = models.ForeignKey(SourceRSS, on_delete=models.CASCADE)
```

Výpis 3: Model definující ukládaná data každého článku

5.4 Přidávání zdrojů a filtrů

K tomu, aby systém mohl začít shlukovat zprávy, byla potřeba přidat do databáze informace o zdrojích, ze kterých by si mohl tato data začít stahovat. Takovéto zdroje by bylo zajisté možné definovat přímo ve zdrojovém kódu, avšak to by pak přišel celý systém o určitou dávku

dynamičnosti a flexibility. Musel jsem tedy stránku s nastavením logicky propojit s aplikací, a to pomocí kontroleru.

O kontrolu veškeré interakce s webovou stránkou se stará soubor **views.py**. Bývá zvykem, že každá vytvořená aplikace Djanga si spravuje tento svůj soubor sama. Já se taktéž vydal touto cestou, ačkoliv by bylo možné využít pouze jeden kontroler, který by spravoval celou aplikaci. To by však v budoucnu při rozšíření projektu mohlo vést ke špatné čitelnosti a přehlednosti kódu. V tomto souboru je tedy definován způsob, jakým má stránka reagovat na určitý požadavek uživatele. Existují dva základní typy těchto požadavků, a to POST a GET. Pokud bychom tuto situaci popisovali z pohledu interakce s uživatelem, tak POST požadavek posílá uživatelsky zadaná data na server očekávajíc konkrétní výsledek. Typicky se jedná například o odeslání registračního či přihlašovacího formuláře. Server následně tento požadavek zpracuje a uživatele přihlásí. Metodou GET se pak uživatel pouze dožaduje určité stránky bez jakýchkoliv přidaných dat. Typicky při jednoduchém procházení webových stránek.

Aby tedy bylo možné přidávat zdroje dynamicky na webové stránce s nastavením, byla potřeba pro každou akci vytvořit odpovídající reakci. V mém konkrétním případě se každá akce změny v nastavení, ať už se jedná o přidání, odebrání či změnu informace, označuje jako metoda POST, jelikož při těchto požadavcích chceme měnit serverová data. Celkem jsem tedy vytvořil a definoval 5 POST metod. Dvě na přidání stránky a zdroje, dvě na odebrání stránky a zdroje a poslední na zpracování formuláře, který řeší nastavení filtrů a informace o zdroji. Po zprovoznění veškeré funkcionality na straně webové stránky i systémové logiky jsem mohl dynamicky za běhu webové aplikace přidat jakékoliv weby, ze kterých jsem chtěl zprávy stahovat. Všechny mnou ručně přidané weby jsou vypsány v tabulce 1. Rozhodl jsem se pro následujících dvacet, jelikož se jedná o relativně známé a často používané české servery, které frekventovaně aktualizují své zprávy a RSS kanály.

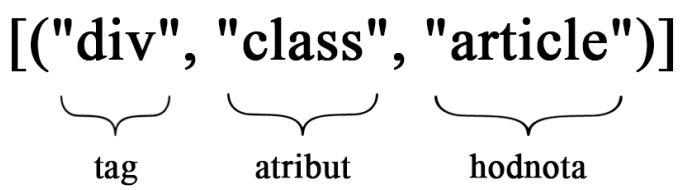
Tabulka 1: Využité internetové zdroje zpráv

novinky.cz	idnes.cz
ihned.cz	lidovky.cz
metro.cz	sport.cz
tyden.cz	aktualne.cz
info.cz	reflex.cz
lupa.cz	zive.cz
e.15.cz	doupe.cz
root.cz	eurozpravy.cz
tn.nova.cz	techarena.cz
svethardware.cz	mobilmania.cz

Pro každý zdroj bylo následně nutné ručně najít jeho RSS kanál a vytvořit filtr, který by ořezal nepotřebné části stažené webové stránky tak, aby zůstal pouze článek a relevantní data.

Ačkoliv měla většina z těchto stránek pouze jeden RSS kanál, tak se stále našlo několik dalších, které měly takovýchto kanálů více a rozhodl jsem se, že v těchto situacích přidám do systému akorát dva z celkového počtu, jelikož mi to pro testovací účely přišlo jako dostatečné. Následně jsem musel pro každý tento kanál vytvořit vlastní filtr, který by ořezával pouze požadované elementy ze článků.

Aby mohl systém články filtrovat, musel v první řadě vymyslet určitá pravidla, která by filtraci umožňovala. Vytvořil jsem tedy vlastní syntaxi pravidel, které tuto filtraci řeší s využitím elementů jazyka HTML. Tento jazyk je využívaný pro zobrazování webových stránek a pomocí značek vytváří hierarchii celé stránky. Obsah článků bývá typicky schovaný v jediném elementu, resp. značce. Tento element v sobě však schovává i další elementy, jako například nadpisek, obrázek, nebo samotný text. Stačí tedy najít pouze rodiče a všechny ostatní chtěné elementy získáme také. Syntaxe mého filtru je tedy poměrně jednoduchá.



Obrázek 6: Syntaxe pravidel pro filtrování

Celý filtr je definován mezi dvěma hranatými závorkami, mezi které již vkládáme požadovaná pravidla. Každé pravidlo začíná a končí kulatou závorkou. Pravidel může být více za sebou, avšak na obrázku 6 je pouze jediné, za které by bylo možné po oddělení čárkou přidat pravidla další. Prvním parametrem pro každé pravidlo je tag onoho rodičovského elementu, který chceme použít k filtraci. Avšak vzhledem k tomu, že na stránce může být vícero tagů se stejným názvem, tak je potřeba blíže specifikovat konkrétního rodiče, kterého chceme skutečně filtrovat. Toto řeší následující dvě hodnoty, tedy atribut a hodnota onoho atributu. Každá značka může obsahovat atribut, který ji jakýmsi způsobem blíže identifikuje. Může se jednat o jedinečný identifikátor, či pouze o jakési skupinové rozeznání. V případě obrázku 6 hledá pravidlo element s tagem „div“, který má atribut „class“, a jejíž hodnota je „article“. Elementů s takovými kritérii může být více a je tedy důležité specifikovat toto pravidlo co nejpřesněji.

S využitím výše zmíněné syntaxe jsem následně vytvořil celkem dva filtry, které se starají o celkové ořezávání článku ze stažených stránek. Práce prvního filtru spočívá v získání těch elementů, které obsahují celý článek. Tento element bývá typicky pouze jeden. Může však nastat i situace, kdy je takových elementů několik a v tom případě je potřeba přidat do tohoto filtru pravidel více. Druhé pravidlo následně řeší ořezávání nechtěných elementů z již vyfiltrovaných elementů obsahující článek. Stává se totiž velmi často, že některé elementy obsažené v rodičovském elementu článku nejsou pro systém chtěné. Může se například stát, že na konci celého článku bývá určitý prvek obsahující reklamu. Díky tomuto druhému filtru se můžeme

velmi jednoduše tohoto irelevantního elementu zbavit. Vytvořil jsem taktéž určitou zkrácenou formu zápisu těchto filtrů, kdy není potřeba vypisovat veškeré elementy, ale stačí zadat pouze tag nebo atribut s hodnotou. Tato pravidla jsou podrobněji popsána na stránce s nastavením a při vytváření filtrů je tedy velmi jednoduché si své nově vytvořené pravidlo zkontrolovat.

5.5 Tvorba modulu pro stahování dat

Pokud jsou zdroje dat nastaveny, může začít proces zpracování RSS kanálů a následné stahování, resp. scrappování, webové stránky. Systém však v budoucnu může stahovat věcí vícero, a proto by bylo vhodné vytvořit jednotný způsob stahování veškerých potřebných dat z internetu. Pro zajištění tohoto jednotného způsobu stahování dat jsem vytvořil modul s názvem **web_manager.py**, který má za úkol obstarávat veškerou komunikaci s internetem a webovými stránkami.

Existuje několik způsobů, jak v Pythonu stahovat webovou stránku. Na výběr je celá řada knihoven, které tuto problematiku řeší. Já jsem se rozhodl využít knihovny requests, což je elegantní a jednoduchá HTTP knihovna pro jazyk Python [13]. Tato knihovna mi byla velkým pomocníkem, jelikož mi umožnila s minimálním počtem řádků kódu stahovat jakýkoliv web.

```
import requests

def load_website(web_url):
    req = requests.get(web_url)
    return req.text
```

Výpis 4: Stažení webové stránky pomocí knihovny requests

Při takovéto základní a jednoduché implementaci jako ve výpisu 4 však může vzniknout řada nevhodných, či jinak nepředvídatelných chování. Může například nastat situace, kdy bude vstupní adresa špatně zadaná, případně nebude systém připojen k internetu a mohlo by dojít k určitým výjimkám. Taktéž si nemůžeme být jisti, zdali bude tato knihovna načítat stránku vždy ve správném formátování a systém by mohl začít pracovat s formátem ISO-8859-1 jako s UTF-8, což by mohlo vést k určitým komplikacím a nežádoucím výsledkům. Všechny tyto zmíněné problémy jsem tedy musel ve svém modulu vyřešit. V případě, že se posléze vyskytne jakákoliv chyba, tak metoda vrátí „None“. Systém tedy bude moci na tuto chybu reagovat a případné stahování zopakuje nebo rovnou přeskočí.

5.6 Metoda parsování značkovacích jazyků

Ve chvíli, kdy se dostaneme k datům zdroje, ať už se jedná o RSS kanál ve formátu XML nebo klasickou webovou stránku ve formátu HTML, tak je musíme nějakým stylem zpracovat do strojově pochopitelného formátu. Díky tomu, že se jedná o jazyky standardizované, tak můžeme

využít několika knihoven, které se zabývají překladem značek takovýchto jazyků do objektů, se kterými můžeme v kódu jednoduše pracovat a manipulovat.

5.6.1 Knihovna Beautiful Soup

K parsování značkovacích jazyků XML a HTML slouží knihovna Beautiful Soup. Jedná se o knihovnu napsanou v Pythonu a umožňuje velmi jednoduché možnosti prohledávání, navigace a manipulace s již zmíněnými značkovacími jazyky [14]. V základním nastavení však využívá základní Python HTML parser, který z mého pozorování nebyl až tak dobře kompatibilní s jazykem XML a občas nedokázal korektně přečíst dokument v tomto jazyce. Beautiful Soup však nabízí i možnost využití vlastního externího parseru, což je taktéž doporučeno v dokumentaci, kde autoři zmiňují několik preferovaných parserů. Já jsem se rozhodl využít parser LXML, který se mi jevil jako ideální volba, jelikož je soustředěn právě a XML a HTML. Jedná se o knihovnu napsanou v jazyce C, takže je i velmi rychlá [15]. V kombinaci těchto dvou knihoven jsem byl již schopen velmi efektivně a korektně zpracovávat veškerá stažená data. Při parsování je však nutné rozlišit, zdali se jedná o dokument typu XML nebo HTML. Tento rozdíl lze pozorovat ve výpisu 5, kde je druhý vstupní parametr funkce pro každý typ jiný.

```
from bs4 import BeautifulSoup

# Parsování XML dokumentu do proměnné xml
xml = BeautifulSoup(xml_website, 'lxml-xml')

# Parsování HTML dokumentu do proměnné html
html = BeautifulSoup(html_website, 'lxml')
```

Výpis 5: Ukázka parsování dokumentů pomocí knihovny bs4 a lxml

5.7 Stahování zpráv

V okamžiku, kdy je nastaven alespoň jeden webový zdroj dat, ze kterého můžeme zprávy stahovat, dokáže systém začít agregovat, resp. scrappovat, všechny zprávy, které najde v RSS kanálu daného zdroje. Celý tento proces je však potřeba spustit. K tomu je určeno zelené tlačítko „Start scrapping“ v pravém horním rohu aplikace. Algoritmus si následně začne v cyklu stahovat všechny RSS kanály a začne procházet elementy <item>, ze kterých si dokáže získat titulek, popisek článku a jeho odkaz. Tyto tři prvky jsou v RSS formátu povinné a můžeme si být tedy jisti, že je algoritmus dokáže vždy načíst s nenulovou hodnotou [9]. Existuje taktéž několik dalších, nyní již dobrovolných elementů, které mohou být obsaženy v RSS kanálu, např. náhledový obrázek, datum publikování, jazyk či kategorie. Z mého pozorování jsem však zjistil, že většina webových stránek poskytuje pouhé minimum, které takový kanál musí obsahovat. Spoléhat se tedy na získání náhledového obrázku pouze z RSS dokumentu není zrovna ideální řešení. Musel

jsem tedy najít takový způsob, u kterého bych si mohl být téměř jistý, že se náhledový obrázek zobrazí uživateli téměř vždy. Tento problém jsem vyřešil s využitím již staženého dokumentu a budu o něm psát o něco později v této kapitole.

Systém tedy využije odkazů získaných z RSS kanálu a postupně se pomocí nich pokusí stáhnout všechny webové stránky. Před samotným začátkem stahování se však nejprve zkontroluje, zdali už článek se stejným odkazem v databázi existuje. V případě, že se článek již v databázi nachází, tak na to systém zareaguje a stahování přeskočí, respektive ani nezačne. Pokud se článek v databázi nenachází, tak začne stahování. Zde existují dva scénáře. Je možné, že se takovéto stahování nepovede a systém zaznamená chybu. V takovém případě se aktuální stahování jednoduše přeskočí. Toto se stává převážně v situacích, kdy některé novinové portály limitují maximální počet načítaných zpráv za určitou jimi definovanou dobu. Pokud má tedy portál nastavený limit 20 načtených stránek za minutu a jejich RSS kanál obsahuje 50 položek, systém dokáže načíst stránek pouze 20 a zbytek bude nucen přeskočit. Pokusí se je však načíst v dalším stahovacím cyklu. V případě úspěšného stažení webové stránky se vytvoří databázový objekt, který ponese veškerá data zprávy. Ze začátku se uloží pouze titulek, odkaz na zprávu a její popisek. Následně se začne zpracovávat samotná stažená webová stránka. Ta se pomocí knihovny Beautiful Soup převede do objektu, ve kterém je možné vyhledávat konkrétní požadované elementy.

```
def load_article(link):
    # Stažení webové stránky pomocí vlastní knihovny web_manager
    website_raw = wm.load_website(link)

    if website_raw is None:
        return

    # Konvertování webové stránky do objektu BeautifulSoup
    web_soup = BeautifulSoup(website_raw, 'lxml')

    # Získání náhledového obrázku z <meta> tagu
    web_thumbnail_url = web_soup.find('meta', property='og:image')
```

Výpis 6: Ukázka stažení webové stránky a získání náhledového obrázku

Následně jsem vyřešil problém s náhledovými obrázky popsanými výše. Zjistil jsem totiž, že valná většina webů využívá pro své články meta tagy, které obsahují veškeré informace z původního RSS kanálu, a to včetně náhledového obrázku. Jedná se konkrétně o elementy využívající takzvaného Open Graph Protocolu [16]. Tento protokol byl původně vytvořen společností Facebook za účelem umožnit integraci webových stránek třetích stran s Facebookem. Díky tomuto protokolu můžeme získávat nadpisy, popisy a náhledové obrázky různých internetových článku z několika tagů. Protokol OGP taktéž využívají jiní softwaroví giganti, jako například Google či Wordpress.

Systém tedy do původního databázové objektu přidá i odkaz náhledového obrázku, jehož získání lze pozorovat ve výpisu 6. Následuje část, kdy se ořežou ze staženého dokumentu veškeré irelevantní a nežádoucí elementy. Typicky se jedná o scripty a kaskádové styly. Samotná implementace takovéhoto ořezu je vyobrazena ve výpisu 7. Tyto elementy by mohly způsobovat potíže při načtení do vlastního webu, jelikož je velká pravděpodobnost, že by se styly obou stránek začaly navzájem přebíjet a vznikla by chaoticky vypadající stránka. Totéž platí pro scripty, které by mohly nevhodně modifikovat určité elementy stránky.

```
def clear_soup(soup):
    [x.extract() for x in soup.find_all('script')]
    [x.extract() for x in soup.find_all('style')]
    [x.extract() for x in soup.find_all('noscript')]
    return soup
```

Výpis 7: Kód řešící extrakci nežádoucích elementů

Dalším krokem je tedy uložení takto očištěné stránky do samotné databáze. Pokud by takovýto obsah neobsahoval obrázky, mohli bychom jej bez problému uložit, avšak tím, že se pokoušíme získat obrázky z elementů cizích stránek, se dostáváme k jedné nepříjemné anomálii. Každý <img> tag obsahuje URL obrázku v atributu *src*. Tento odkaz však nemusí být kompletní. Pokud bychom jako autoři smyšleného webu „stránka.cz“ chtěli zobrazit fotku na naší stránce pomocí odkazu v tagu <img>, naskytly by se nám dvě možnosti. Tou první by bylo přidat odkaz celé absolutní cesty, např. „stránka.cz/obrázek.jpg“. Toto je ideální stav, jelikož kdokoliv, kdo by využil takovéhoto linku by se dostal k požadovanému obrázku. Existuje však i druhá varianta, při které stačí obrázku předat pouze relativní cestu, např. „/obrázek.jpg“. V takovémto případě bude obrázek v pořádku načten, pokud se nachází na webu „stránka.cz“, avšak v případě využití stejné cesty v naší webové stránce „našestránka.cz“ se nezobrazí nic, jelikož na odkazu „našestránka.cz/obrázek.jpg“ neexistuje žádný soubor.

Vytvořil jsem tedy kontrolu URL odkazu v každém <img> tagu. Algoritmus si pomocí knihovny tld zjistí, zdali odkaz obsahuje absolutní či relativní cestu a pokud se jedná o cestu relativní, přidá jí odkaz na doménu webu, ze kterého obrázek pochází a tím zaručí správné načtení obrázku pomocí absolutní cesty [17]. Po extrakci nežádoucích elementů a úpravě všech obrázkových URL je textový stav stránky přidán do databázového objektu. Původně jsem zamýšlel, že by se tato stránka před uložením ještě vyfiltrovala pomocí filtru daného zdroje a až poté uložila do databáze, což by z jisté části zmenšilo velikost ukládaných dat. Tento přístup jsem však zvolil až po odladění filtrů. Tím, že jsem prvně ukládal články nefiltrované, jsem mohl upravovat filtry za běhu a vidět výsledky jejich změn okamžitě, a to na již stažených stránkách.

Následně zbýval už jen poslední proces, a to extrakce klíčových slov pro budoucí vyhledávání podobných dat. Pro tuto část jsem se rozhodl použít knihovnu rake-nltk, neboli Rapid Automatic Keyword Extraction, taktéž známou pod zkratkou RAKE [18]. Tato knihovna umožňuje extrahovat klíčová slova z textů o různých délkách. Pro čistou extrakci je však nejprve potřeba získat

samotný článek bez jakýchkoliv nežádoucích elementů. Stránka se tedy ořeže pomocí již zmíněných filtrů a algoritmus pro extrakci může začít dělat svoji práci. Celý proces extrakce klíčových slov lze pozorovat ve výpisu 8. Vzhledem k tomu, že český jazyk není pro tuto knihovnu podporovaný, jsem byl pro zlepšení výsledku donucen nastavit extrakci relevantních klíčových slov na maximální délku jedna. Systém byl ale překvapivě i tak docela přesný ve výběru relevantních a důležitých slov.

```
def get_keywords_from_text(text):
    r = Rake(min_length=1, max_length=1)
    r.extract_keywords_from_text(text)
    return r.get_ranked_phrases_with_scores()
```

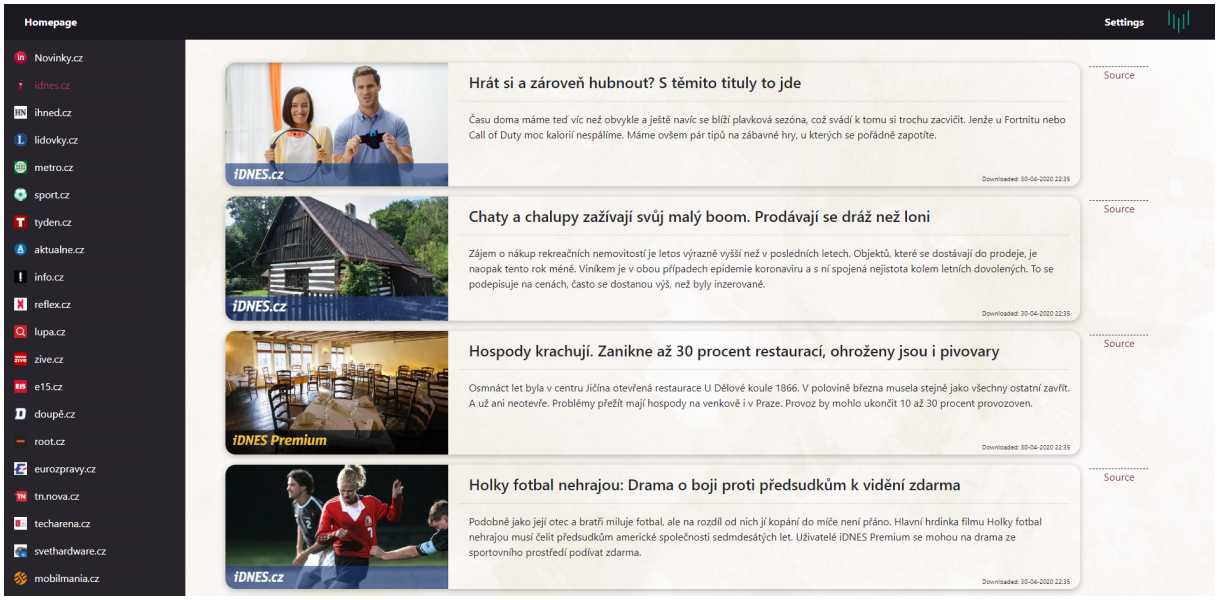
Výpis 8: Metoda pro extrakci klíčových slov pomocí knihovny rake-ntlk

Výstup těchto klíčových slov je pak převeden do formátu JSON, který je uložen jakožto poslední informace do databázového objektu. Ten je pak pomocí metody save() uložen z paměti do samotné SQLite databáze na disku.

5.8 Zobrazení zpráv

Při otevření úvodní obrazovky hlavní stránky se zobrazí v levém menu seznam všech již přidaných zdrojů. Defaultně je vybrán zdroj první. Systém si načte posledních 10 uložených zpráv stažených ze stránek daného zdroje z databáze a pošle je k zobrazení uživateli. Takovouto plně načtenou stránku je možné vidět na obrázku 7. U každé zprávy se však zobrazuje pouze titulek, popis zprávy a náhledový obrázek. Uživatel nepotřebuje vidět veškerý obsah článku, jelikož je dost možné, že si bude chtít přečíst třeba jen jednu zprávu z celého seznamu. Načítání celého článku probíhá tedy až při jeho samotném výběru. Do té doby má uživatel k dispozici pouze rychlý seznamový přehled o všech posledních zprávách.

Při kliknutí na požadovanou zprávu se uživateli začne načítat kompletní článek. Zpráva je tedy vyfiltrovaná tak, aby mohla být přímo vložena do stránky ve stejné podobě, jako je pomocí JS načtena. Mnou vytvořené CSS se následně postarají o vzhled tohoto článku, aby byl pro uživatele i snadno čitelný. Systém posléze odešle AJAX požadavek, který si vyžádá další podobné články, které jsou poté zobrazeny po pravé straně od samotného článku. Na obrázku 8 je možno vidět vzhled plně načtené stránky se článkem a podobným obsahem.



Obrázek 7: Seznam nejnovějších zpráv na hlavní stránce

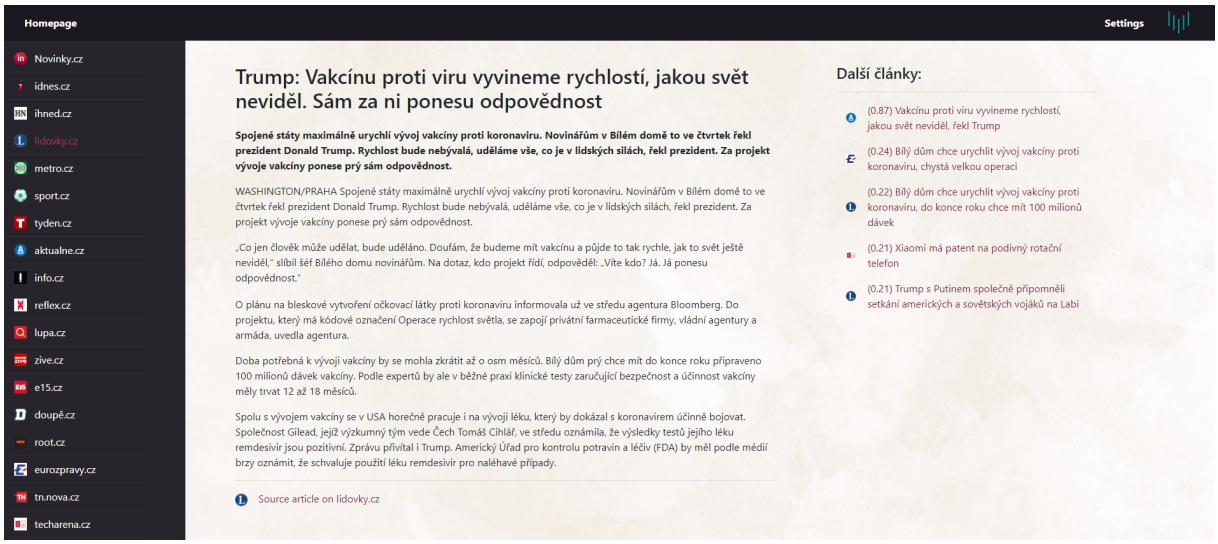
5.9 Články s podobným obsahem

Pro každý článek je možné nalézt články s podobným obsahem. Díky tomu, že je při ukládání, resp. scrappování, článku vytvořen seznam klíčových slov daného článku, můžu využít vlastní logiky a pokusit se pomocí nich nalézt podobný článek obsahující co nejvíce stejných klíčových slov. Každé takové slovo má však také i svou váhu, resp. hodnotu relevance ve článku, ze kterého bylo získáno. Začal jsem tedy porovnávat klíčová slova aktuálně zobrazeného článku se všemi ostatními a počítal, kolik z nich je stejných a jaká je jejich celková váha. Podle jejich četnosti a vah jsem následně vypočítal koeficient shody, pomocí kterého systém rozhodne, který článek je aktuálnímu článku podobný více a který méně. Pro tento výpočet je potřeba pouze dvou proměnných. Do první proměnné uložím součet vah všech klíčových slov obou článků. Druhá proměnná bude obsahovat taktéž součet vah klíčových slov, tentokrát však pouze těch slov, která jsou obsažena v obou článcích. Tyto dvě hodnoty mezi sebou poté vydělím dle vzorečku (1) a získám koeficient shody, pomocí kterého můžu jednoduše usoudit jejich podobnost.

$$\frac{Váha\ stejných\ slov}{Váha\ všech\ slov} = Koeficient\ shody \tag{1}$$

Takovéto porovnání by však při velkém počtu zpráv bylo velice zdlouhavé. Rozhodl jsem se tedy, že bude systém vyhledávat podobné zprávy pouze s rozdílem maximálně jednoho týdne od publikování oné zprávy. Toto hledání zajistí velmi rychlou odpověď s dostatečně relevantními výsledky. Pokud bych se však na tuto problematiku podíval z pohledu velkých firem, jejichž agregátor by byl určen pro produkční prostředí, tak by byl tento způsob v podstatě neakcep-

tovatelný. Systém by zažíval neustálé hardwarové zatížení, jelikož by při každé otevřené zprávě musel znovu prohledávat veškeré zprávy podobného obsahu, a to i v případě, že je už někdy předtím našel. Tento problém by se dal vyřešit indexací. Při každém stahování zpráv by systém jednorázově prošel veškeré články v definovaném časovém období a uložil, resp. zaindexoval, všechny, kterým je stahovaný článek podobný. Pokud by následně koncový uživatel zobrazil jakoukoliv zprávu, všechny zprávy s podobným obsahem by byly okamžitě k dispozici. Já se však pro testovací účely rozhodl využít možnost načítání podobného obsahu až při zobrazení samotného článku. Výsledek je v konečném důsledku stejný, akorát s menším zpožděním. Pokud by se vše indexovalo, tak by stahování všech zpráv trvalo mnohonásobně déle.



Obrázek 8: Stránka se článkem a nabídkou podobného obsahu

5.10 Paralelizace

Ačkoliv systém dělal přesně to, co má, tak rychlost stahování stránek nebyla zrovna ideální. Tím, že se kód provádí řádek po řádku, vniká jistá řada nevýhod. V mém případě se jednalo o to, že při stahování stránek byl celý systém zmražen, resp. zaseknutý, a uživatel nemohl se stránkou nijak interagovat do doby, než byly všechny zprávy staženy. Další problém se vyskytl v rychlosti stahování stránek. Systém nedokázal efektivně stahovat více než jednu stránku najednou. V případě velkého objemu stahovaných dat bylo nutné čekat opravdu velmi dlouhou dobu, než se všechna data stáhnou a uloží.

Toto vše vyřešila paralelizace systému pomocí vláken. Python 3 nabízí vysokoúrovňové rozhraní spravující systém vláken, anglicky threads. Využití této knihovny mi umožnilo vyřešit několik nepříjemností najednou. Vytvořil jsem tedy samostatné vlákno, které bylo zodpovědné za stahování veškerých zpráv. Díky tomu je hlavní vlákno stále dostupné a uživatel může interagovat s celou stránkou pouze s minimálním omezením i při aktivním stahování nových zpráv.

Využití pouze jediného dalšího vlákna by však stále znamenalo, že systém může stahovat pouze jednu zprávu v konkrétní čas. Využil jsem tedy i možnosti vnoření několika vláken do jiného vlákna a vytvořil pro každé stahování vlákno samostatné. Systém tedy pracuje následovně. Spustí se sekundární vlákno, na kterém poběží veškeré procesy týkající se stahování nových zpráv. Toto rodičovské vlákno začne postupně iterovat všechny zdroje dat a začne si načítat jejich RSS kanál. Každý zdroj reprezentuje jednu iteraci v cyklu rodičovského vlákna. V každé této iteraci se najednou vytvoří tolik vláken, kolik je článků v RSS kanálu. V každém vláknu se následně spustí stahování jen jedné stránky z mnoha. Poté systém vyčká na ukončení všech těchto stahovacích vláken a následně může pokračovat v další iteraci. Celý proces stahování je poté zopakován po určité časové pauze a běží v nekonečné smyčce jediného vlákna.

5.11 Výkon aplikace

Systém prošel při své implementaci několika důležitými změnami, které se silně podepsaly na jeho celkovém výkonu, rychlosti a efektivitě. Rozhodl jsem se tedy vytvořit i statistiky týkající se výkonnostních rozdílů při odlišných implementačních metodách různých částí systému. Jako první jsem změřil rychlost stahování zpráv bez paralelizace a s paralelizací při využití SQLite. Následně jsem celý test zopakoval s použitím databáze PostgreSQL, abych zjistil, jak moc jsou tyto databáze rozdílné. Všechna měření byla provedena na počítači s procesorem Intel Core i7-4770 o frekvenci 3.40 GHz a internetovým připojením s rychlostí stahování 300 Mb/s.

Tabulka 2: Rychlost stahování s využitím SQLite

Zdroj dat	Počet článků	bez paralelizace	s paralelizací
Novinky.cz	100	66,7s	25,6s
Idnes.cz	30	16,9s	7,8s
Ihned.cz	150	91,4s	27,7s
Lidovky.cz	30	15,6s	8,7s
Průměrná rychlost stažení 1 článku		0,59s	0,248s

Tabulka 3: Rychlost stahování s využitím PostgreSQL

Zdroj dat	Počet článků	bez paralelizace	s paralelizací
Novinky.cz	100	56,3s	18,3s
Idnes.cz	30	14,4s	7,9s
Ihned.cz	150	80,2s	22,0s
Lidovky.cz	30	12,5s	6,5s
Průměrná rychlost stažení 1 článku		0,5s	0,2s

Dle výsledků je krásně vidět, jak dokázala paralelizace až několikanásobně urychlit celý proces agregace zpráv. V případě velkého množství stažených článků ze zdroje Ihned.cz v tabulce 3 se jednalo až o 3,5násobné urychlení. A to vše při stahování pouze z jednoho zdroje dat, což mohlo také zpomalit celkové stahování. Pokud by bylo stahování rovnoměrně rozděleno mezi vícero zdrojů v jediný okamžik, mohli bychom pravděpodobně dosáhnout ještě lepších výsledků. Rozdíl mezi oběma databázemi byl taktéž docela znát. Pokud mezi sebou porovnám jejich rychlosti, tak se v případě využití PostgreSQL stále jedná o úctyhodné zrychlení a výsledky zde podporují i teoretické znalosti, tedy že databáze využívající běžícího serverového procesu je rychlejší a efektivnější než jediný statický soubor jako u SQLite. Rychlostní rozdíl se zde pohybuje zhruba v rozmezí 20 až 25 procent, což je sice stále dost velký výkonnostní nárůst, avšak oproti paralelizaci ne až tak značný.

Rychlost stahování zpráv však není jediná důležitá hodnota a neměli bychom zapomínat i na velikost dat uložených na diskovém uložišti. Jak jsem již zmínil dříve, stránky se při testování před uložením ořezávaly pouze o skripty a CSS. Samotné filtrování probíhalo až při zobrazování článku, a to z důvodu, aby se dalo tyto filtry jednoduše modifikovat bez potřeby opětovného stahování stránek. Po odladění filtrů jsem si zapsal velikost databáze a počet již stažených článků. Následně jsem všechny stránky v databázi vyfiltroval pomocí mnou vytvořených filtrů a podíval se na rozdíl ve velikostech před a po. Z tabulky 4 je vidět, že samotné filtry dokázaly snížit objem dat až téměř 10násobně, což je dost znatelný rozdíl.

Tabulka 4: Rozdíl velikosti filtrovaných a nefiltrovaných dat

	Velikost 12694 záznamů	Průměrná velikost záznamu
Nefiltrovaná data	1 029 100 kB	81 kB
Filtrovaná data	108 516 kB	8,5 kB

Také bych zde rád zmínil i kvalitu a rychlost hledání podobných článků. Při zobrazení některé z posledních stažených zpráv bylo v případě využití zdrojů z tabulky 1 prohledáváno zhruba dalších 4000 článků na podobné téma. Tento proces trval na mém počítači s procesorem Intel Core i7-4770 průměrně 500 milisekund. Pokud by systém využíval pouze jednotlivec, tak se, dle mého názoru, jedná o docela dobrou hodnotu, která nijak nekazí dojem celé aplikace. V případě využití systému několika lidmi zároveň by však byla tato rychlost nejspíše nedostatečná a musel by se celý proces vyhledávání urychlit. Podobnost článků se také velmi liší a významnou roli zde hraje délka článku a jeho téma. Systém určuje podobnost hodnotou mezi nulou a jedničkou, přičemž jednička je nevětší možná podobnost. U delších článků sice můžeme porovnávat slov více, ale zároveň je i menší šance, že budou tyto shody relevantní. Jejich hodnocení proto bývá typicky o něco menší než u článků kratších až průměrně dlouhých. Dle mého pozorování jsem vyhodnotil, že články pod hodnocení 0.5 bývají podobné zřídka, mezi 0.5 a 0.75 často a nad 0.75 téměř vždy.

6 Závěr

Tato bakalářská práce byla věnována tématu agregace internetových zpráv. Vysvětlil jsem, k čemu agregátor zpráv vlastně slouží a taktéž popsal několik různých řešení, která jsou ve světe ke shlukování zpráv využívána. V teoretické části této práce jsem zmínil technologie, které se dají použít pro práci s daty, tvorbu webových stránek a agregaci zpráv z internetu. Následně jsem v další kapitole vytvořil analýzu systému, jenž by dokázal fungovat jakožto webový agregátor internetových zpráv. Poté jsem se za pomoci svých znalostí pokusil takový systém sestavit. Vytvořil jsem webovou aplikaci, která shlukuje články z internetových stránek, zobrazí je uživateli a nabídne několik zpráv na podobné téma. Práci jsem završil otestováním funkcionality tohoto systému za několikaměsíční časové období na 20 českých datových zdrojích a popsal jeho výkonnostní charakteristiky.

Systém jako takový je funkční, avšak algoritmus pro hledání podobných článků není úplně přesný a nachází se zde spousta prostoru pro zlepšení. Výhodou tohoto systému je však jeho efektivita při poměrně nenáročné implementaci, a to hlavně díky již existujícím knihovnám. V budoucnu by se tento projekt dal rozšířit s využitím umělé inteligence a indexací klíčových slov. Tvorbou této bakalářské práce jsem si zároveň zdokonalil své znalosti jazyka Python, frameworku Django a několika dalších webových technologií.

Literatura

1. TATRANSKÁ, Martina. *Webová agregace zpráv : současné světové trendy* [online]. Praha, 2006 [cit. 2020-04-01]. Dostupné z: https://is.cuni.cz/webapps/zzp/detail/27566/. Diplomová práce. Univerzita Karlova v Praze.
2. GINGRAS, Richard. *A look at how news at Google works* [online] [cit. 2020-04-01]. Dostupné z: https://www.blog.google/products/news/look-how-news-google-works/.
3. PRIORA, Giulia. *News Aggregation and the Reform of EU Copyright Law* [online] [cit. 2020-04-01]. Dostupné z: https://cmds.ceu.edu/article/2018-07-03/news-aggregation-and-reform-eu-copyright-law.
4. *Python 3.6.10 documentation* [online] [cit. 2020-04-01]. Dostupné z: https://docs.python.org/3.6/.
5. *Django. The Web framework for perfectionists with deadlines* [online] [cit. 2020-04-01]. Dostupné z: https://www.djangoproject.com/.
6. OLSZOWSKI, Pavel. *Dotazovací jazyk SQL* [online] [cit. 2020-04-01]. Dostupné z: http://books.fs.vsb.cz/SQLReference/.
7. *SQLite* [online] [cit. 2020-04-01]. Dostupné z: https://www.sqlite.org/index.html.
8. *PostgreSQL* [online] [cit. 2020-04-01]. Dostupné z: https://www.postgresql.org/about/.
9. *Jak na internet - RSS kanály* [online] [cit. 2020-04-01]. Dostupné z: https://www.jaknainternet.cz/page/1640/rss-kanaly/.
10. BUREŠ, Jiří. *Interval - RSS 2.0* [online] [cit. 2020-04-01]. Dostupné z: https://www.interval.cz/clanky/rss-20/.
11. *Bootstrap - Introduction* [online] [cit. 2020-04-01]. Dostupné z: https://getbootstrap.com/docs/4.4/.
12. *JQuery* [online] [cit. 2020-04-01]. Dostupné z: https://jquery.com/.
13. *Requests: HTTP for Humans* [online] [cit. 2020-04-01]. Dostupné z: https://requests.readthedocs.io/en/master/.
14. *Beautiful Soup Documentation* [online] [cit. 2020-04-01]. Dostupné z: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.
15. *lxml - XML and HTML with Python* [online] [cit. 2020-04-01]. Dostupné z: https://lxml.de/.
16. *The Open Graph Protocol* [online] [cit. 2020-04-01]. Dostupné z: https://ogp.me/.
17. *Balíček tld* [online] [cit. 2020-04-01]. Dostupné z: https://pypi.org/project/tld/.
18. *Balíček rake-ntlk* [online] [cit. 2020-04-01]. Dostupné z: https://pypi.org/project/rake-nltk/.